The Hong Kong Polytechnic University
Department of Electronic and Information Engineering

Subject: EIE522 Pattern Recognition

Lab 1: Statistical Pattern Recognition

Objective:

In this laboratory exercise, you will use MatLab to study the distribution of 1D- and 2D-training samples from a population of several classes. You will plot the probability density function for each of the classes and design the optimal Bayes classifiers based on the training samples.

Introduction:

In a pattern recognition problem, we have a number of classes and we want to assign a query input to one of these classes. Statistical pattern recognition employs statistical properties to describe the samples of each class. Usually, the distribution of the samples in a class is assumed to be governed by the Gaussian density function. For 1D feature samples, the probability density function (pdf) is given as follows:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{ -\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2 \right\}, \tag{1}$$

where x represents a sample, μ the mean, and σ the variance of the density function. If we know that the samples belong to the class i, which is denoted as ω_i, the pdf is then represented by a conditional probability function as follows:

$$p(x \mid \omega_i) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left\{ -\frac{1}{2}\left(\frac{x-\mu_i}{\sigma_i}\right)^2 \right\}. \tag{2}$$

If the dimension of the samples is higher than one, i.e. multi-dimensional, the class-conditioned pdf is given as follows:

$$p(\mathbf{x} \mid \omega_i) = (2\pi)^{-\frac{n}{2}} \mid \mathbf{\Sigma}_i \mid^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^T \mathbf{\Sigma}_i^{-1} (\mathbf{x}-\boldsymbol{\mu}_i) \right], \tag{3}$$

where $\boldsymbol{\mu}_i$ is the class-specific mean, $\mathbf{\Sigma}_i$ the covariance matrix, and $|\mathbf{\Sigma}_i|$ the determinant of the matrix $\mathbf{\Sigma}_i$. Suppose that the *a priori* class probability or the probability of the occurrence of a class i is denoted as $P(\omega_i)$. The sum of this probability over all the classes is equal to one, i.e.

$$\sum_i P(\omega_i) = 1. \tag{4}$$

The unconditional density function $p(\mathbf{x})$ is then given by

$$p(\mathbf{x}) = \sum_i p(\mathbf{x} \mid \omega_i) P(\omega_i). \tag{5}$$

In a c-class case, a classifier can be designed based on *discriminant functions,* which is denoted as $g_i(\mathbf{x})$, $i = 1, 2, \ldots, c$. These discriminant functions partition the feature space of dimension d, $\Re^d$, into c non-overlapped sub-regions. An input $\mathbf{x}$ will be assigned to class m if

$$g_m(\mathbf{x}) > g_i(\mathbf{x}), \quad \forall\ i = 1, 2, \ldots, c \text{ and } i \neq m. \tag{6}$$

The Bayes classifier is an optimal classifier, which is derived by minimizing the total average loss when we make the decision to assign $\mathbf{x}$ to a class and assuming that the loss is zero when the classification is correct, and one otherwise. The discriminant function for class i is given as follows:

$$g_i(\mathbf{x}) = p(\mathbf{x} \mid \omega_i) P(\omega_i). \tag{7}$$

This discriminant function can also be obtained by considering the *a posteriori* probability or the measurement conditioned probability $P(\omega_i \mid \mathbf{x})$. This is the probability of a measured sample $\mathbf{x}$ belonging to the class ω_i. The input sample $\mathbf{x}$ will be assigned to class i if $P(\omega_i \mid \mathbf{x})$ has the largest value among all the classes. Using Bayes' rule, we have

$$P(\omega_i \mid \mathbf{x}) p(\mathbf{x}) = p(\mathbf{x} \mid \omega_i) P(\omega_i) \quad \text{or} \quad P(\omega_i \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \omega_i) P(\omega_i)}{p(\mathbf{x})}. \tag{8}$$

As $p(\mathbf{x})$ is common to all classes, so we have the discriminant function (7) for the class ω_i.

Provided Files:

Download the zip file “PR_Lab1_StatPR.zip” from the subject web page, and save and decompress the zip file in your work folder. You may find the following files:

1.	*data1d.m*:	a file containing 1D training samples
2.	*data2d.m*:	a file containing 2D training samples
3.	*plotdata1d.m*:	a MATLAB file to plot 1D data
4.	*plotdata2d.m*:	a MATLAB file to plot 2D data
5.	*plotboundary1d.m*:	a MATLAB file to plot 1D boundary
6.	*plotboundary2d.m*:	a MATLAB file to plot 2D boundary
7.	*gendata.m*:	a MATLAB file to generate a data set with multiple classes and samples of any dimension.

Start Your MATLAB:

1. Create a folder for this laboratory exercise. This work folder may be “**C:\MATLAB6p5\work\**”, “**c:\temp\**” or “**d:\temp\**”. Note that you should be able to access, save, and load files in this folder.
2. Start MATLAB: select **Start → Programs → MATLAB 6.5 → MATLAB 6.5**.
3. Add your work folder to MATLAB: in MATLAB, choose **File | Set Path** and add the path of your work folder.

One-dimensional Pattern Space:

1. Load the sample data from the file “**data1d.m**” by typing **data1d** in MATLAB. You may then see 2 items in the MATLAB workspace, which are:
    - Item **x**, which stores the sample values, and
    - Item **y**, which labels the class of the corresponding samples in **x**.

    (a) Find the following information about the training samples: number of classes and number of training samples for each class. It is given that the range of the data is between 0 and 100.

    (b) Plot the normalized histograms of the respective classes by considering (i) one out of every five samples, and (ii) all the samples in the data file. Comment on your results.

    Hints:The following MATLAB functions may help you:

```
close all                       % close all figures
clear all                       % clear all variables
data1d;                         % load the 1-D data from the file “data1d.m”
numClass = ?;                   % number of classes found in 1(a)
numSample = ?;                  % number of samples found in 1(a)

minValue = min(x)               % the minimum value of the data in vector x
maxValue = max(x)               % the maximum value of the data in vector x
count = length(x)               % number of elements in vector x

% This is an example to illustrate the construction of a histogram,
% and one sample is selected out of every 5 samples
step = 5;                       % one out of every 5 samples
for class = 1:numClass,
  % Get the data from one of the classes
  index = find(y == class)      % find the indices of data from the current class
                                % N.B. the index starts at 1 in MATLAB
  xtemp = x(index);             % put the data of the current class into 'xtemp'
  numSampleClass = length(xtemp);         % number of samples in this class
  xtemp2 = xtemp(1 : step : length(xtemp));% Extract one out of every 5 samples
  N(class, :) = hist(xtemp2, [0:100]);    % get the bin values of the histogram
end
figure; bar(N(1,:), 'b'); axis( [0, 100, 0, max(N(1,:))] )  % plot the histogram for class 1
xlabel('Values'); ylabel('Number of elements'); title('Histogram of Class 1')
```

2. Assume that the pdf of each of the classes is also a Gaussian function.

   (a) Estimate $P(\omega_i)$, the mean and the standard deviation of each class.

   (b) Find and plot the pdf for each class. Discuss your results.

   Hints:

- The MATLAB functions for calculating the mean and standard deviation of the values in a vector x are as follows:

```
xmean = mean(x)                % the mean of vector x
xstd = std(x)                  % the standard deviation of vector x
```

- We need to use a **For** loop to find the means and standard deviations of the respective classes. Suppose that **u** and **stddev** are the arrays used to store the means and standard deviations, where **u = [u$_1$, u$_2$, …, u$_{numClass}$]**, and **stddev = [stddev$_1$, …, stddev$_{numClass}$]**. The scalars **u$_1$** and **stddev$_1$** are the mean and standard deviation, respectively, of the first class, and so on. The following code may help you.

```
for class = 1:numClass,           % loop for each class,
                                  % where numClass is the number of classes
  index = find(y == class);       % find the indices of current class's data
  xtemp = x(index);               % put the current class's data into 'xtemp'

  % Find the mean and standard deviation of the data
  u(class) = mean(?);             % the mean; what should be the "?"?
  stddev(class) = std(?);         % the standard deviation; what should be the "?"?
  priori(class) = length(?) / numSample; % P(wi); what should be the "?"?
end
```

- The pdf for class i is $p(x\,|\,\omega_i) = \dfrac{1}{\sqrt{2\pi}\sigma_i}\exp\left\{-\dfrac{1}{2}\left(\dfrac{x-\mu_i}{\sigma_i}\right)^2\right\}$, where μ_i and σ_i are the mean and the standard deviation, respectively. To plot the pdf, you have to generate the data point for x and then compute the corresponding class-conditioned pdf $p(x|\omega_i)$. As the data range from 0 to 100, the following code can generate the data points:

```
numInterval = 100;                   % number of intervals
maxValue = 100; minValue = 0         % minimum and maximum values of the data
% form a vector whose first and last values are minValue and maxValue,
% respectively, and the difference between successive data is
% (maxValue –minValue)/numInterval
xx = minValue : (maxValue –minValue)/numInterval : maxValue;
```

  xx is a vector of size 1×101, so is $p(x|\omega_i)$. Suppose there are numClass classes, then there are numClass 1×101 arrays, i.e. $p(x|\omega_1)$, … and $p(x|\omega_{numClass})$. Here, we will use a numClass ×101 matrix **p** to store all the pdf, where the first row of **p**, **p(1, :)** stores the pdf of the 1st class, the second row of matrix **p**, **p(2, :)** stores the pdf for the 2nd class, etc. The following code may help you.

```
for class = 1:numClass,           % loop for each class
   % Find the probability density function (pdf) for this class
   p(class, : ) = 1 / sqrt(2*pi) / stddev(class) *...
      exp(-0.5 * ((xx - u(class))/stddev(class)).^2);
end
```

- The following code may help you to plot the pdf.

```
figure; plot(xx, p);              % plot the pdf
xlabel('x'); ylabel('p ( x | w_i )'); title('Plot of pdf of 1-D data')
legend(['Class 1'; 'Class 2'; 'Class 3'; 'Class 4'])
```

3. In this part, you investigate the discriminant function for each class.

   (a) Find and plot the discriminant functions of the classes. How do these differ from the pdf of the classes?

   (b) Plot the boundaries on the graph between the classes.

   (c) What is the sum of the discriminant functions of the classes? Discuss your results.

   (d) If $P(\omega_2)$ is increased by 0.1 and the other $P(\omega_i)$ are adjusted so that $\Sigma_i P(\omega_i) = 1$, repeat (b) and discuss your results.

   Hints:

   - The discriminant function for the i^{th} class is $g_i(x) = p(x|\omega_i)\ P(\omega_i)$. Similar to Step 2, a 4×101 matrix, **discriminant**, is used to store the values of the discriminant functions, where each row of the matrix, denoted as **discriminant(i, :)**, stores the values of the discriminant function for the i^{th} class. The following code may help you.

```
for class = 1:numClass,                 % loop for each class
   % Find the discriminant function for this class
   discriminant(class, : ) = p(class, : ) * priori(class);
end
```

   - The following code may help you to plot the discriminant function.

```
fig = figure; plot(xx, discriminant)    % plot the discriminant functions
xlabel('x'); ylabel('g(x)');
title('Plot of Discriminant Functions of 1-D data')
legend(['Class 1'; 'Class 2'; 'Class 3'; 'Class 4'])
```

   - If a sample x belongs to class ω_i, then $g_i(x) > g_j(x)$, where $j = 1, 2, \ldots, numClass;\ j \neq i$. Therefore, to plot the boundaries on the graph of the discriminant functions, we need to find the class whose discriminant function is a maximum for each data point in the array **xx**. You may use the function "**plotboundary1d**" to determine the classes for each point, and so plot the boundary. The following code may help you.

```
[maxDiscri, yy] = max(discriminant);    % determine the choice 'yy'
peak = max(maxDiscri);                  % find the discriminant function with maximum value
figure(fig); hold on                    % call the last figure, and hold it on
plotboundary1d(xx, yy, peak)            % plot the boundaries
hold off
% Sum of areas of the discriminant functions
total_area = sum(sum(discriminant))
```

Two-dimensional Pattern Space:

1. Load the sample data from the file "**data2d.m**" by typing **data2d** in MATLAB. You may see 2 items in the MATLAB workspace, which are:
   - Item **x**, which stores values of data points, and
   - Item **y**, which indicates the class of each data point.

   Plot the graph to show the data with different classes.

   Hints: You may use the function "**plotdata2d**" to plot the graph. This function is used to plot 2D data of different classes, and can also return the approximate ranges of the data for each class. The following codes may help you.

```
close all                               % close all figures
clear all                               % clear all variables
numClass = ?;                           % number of classes, there are ? classes
numDimension = 2;                       % the dimension; it is 2
data2d                                  % load the 2-D data from the file "data2d.m"
figure                                  % prepare to draw a figure, which is the first figure
plotdata2d(x, y);                       % plot the 2-D data of different classes
                                        % and return the approximate data range
xlabel('x1 (Feature 1)'); ylabel('x2 (Feature 2)')
title('2-D data'); legend(['Class 1'; 'Class 2'; 'Class 3'; 'Class 4'])
```

2. Assume that the pdf of each of the classes is also a Gaussian function.

   (a) Estimate $P(\omega_i)$, the mean and the covariance matrix of each class, given that the covariance matrices are $\Sigma_i = E_i\{(\boldsymbol{x} - \boldsymbol{\mu}_i)(\boldsymbol{x} - \boldsymbol{\mu}_i)^T\}$.

   (b) Find and plot the pdf for each class. Discuss your results.

   Hints:

- Suppose there are L classes. A $2\times L$ matrix **u** is used to store the means, where the ith column of the matrix **u**, i.e. **u(:, i)**, stores the mean of the ith class. Similarly, a $2\times 2\times L$ matrix **C** is created to store the covariance matrices, where **C(:, :, i)** stores the covariance matrix of the ith class. The following code may help you, or you can use other methods to find the means and covariance matrices.

```
for class = 1:numClass,              % loop for each class
   index = find(y == class);         % find the indices of data belonging to current class
   xtemp = x(:, index);              % put the data of the class into 'xtemp'
   % Find the mean and covariance matrix of the data
   u(:, class) = mean(xtemp, 2);     % the mean of this class
   matrix = [ ];                     % for initialization
   for n = 1:numDimension,           % loop for each dimension
      matrix(n,:) = xtemp(n, :) - u(n, class);
   end
   Ctemp = matrix * matrix.' / length(xtemp);          % convariance matrix
   C(:,:,class) = Ctemp;          % store the covariance matrix of this class
   priori(class) = length(y(y == class)) / length(y);       % P(wi)
end
```

- Similarly, suppose the ranges of the 2D data are both between 0 and 100. You generate 50×50 data points for each class, and then plot the 2D Gaussian density functions. The following code may help you.

```
numPoints = 50;             % set the number of points
range1(1) = 0; range2(1) = 0;
range1(2) = 100; range2(2) = 100;
x1 = range1(1) : (range1(2) - range1(1))/numPoints : range1(2);
x2 = range2(1) : (range2(2) - range2(1))/numPoints : range2(2);
[x11, x22] = ndgrid(x1, x2);
xx = [x11(:), x22(:)].';    % (:) means transforming a row vector to a column vector
                            % .' means performing transpose, B = transpose(A) <=> B = A.'
```

- The class-conditioned pdf is $p(\boldsymbol{x}\,|\,\omega_i) = (2\pi)^{-\frac{n}{2}}\,|\boldsymbol{\Sigma}_i|^{-\frac{1}{2}}\exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right]$ for a multi-dimensional Gaussian distribution, where $\boldsymbol{\mu}_i$ and Σ_i are the mean and covariance matrix for the ith class. The MATLAB functions "**inv**", "**det**", "**abs**" may be used to implement this equation. Similarly, we use **xx** to compute the pdf. Suppose that **xx** is of size 2×2601, then the size of $p(\boldsymbol{x}|\omega_i)$ is 1×2601. As we are considering a 2D problem, the 1D array of size 1×2601 is converted to a matrix of size 51×51, which is stored as **p(:, :, i)** for the ith class. The following code may help you.

```
for class = 1:numClass,              % loop for each class
   Ctemp = C(:, :, class);           % construct the covariance matrix of this class

   % Find the probability density function (pdf) of this class
   temp = 1 / (2*pi) * (abs(det(Ctemp)))^(-0.5);
   for n = 1:length(xx),             % loop for each data point
      tmp = xx(:, n) - u(:, class);
      ptemp(n) = temp * exp(-0.5 * tmp.' * inv(Ctemp) * tmp);
   end
   % Convert the 1D pdf, 'p', into 2-D
   p(:, :, class) = reshape(ptemp, [length(x1), length(x2)]);
end
```

- The MATLAB function "**surfc**" or "**meshc**" can be used to plot the pdf.

```
peak = max(max(max(p)));              % find the maximum value of the pdf
figure; hold on; grid on              % hold on the 2nd figure
                                      % add major grid lines in the graph
axis([range1, range2, -peak, peak])   % set the axes of the graph
for class = 1:numClass,               % loop for each class
   surfc(x11, x22, p(:, :, class))    % plot the pdf
end
xlabel('x1 (Feature 1)'); ylabel('x2 (Feature 2)'); zlabel(' p ( x | w_i )')
title('Plot of PDF of 2-D data')
hold off; view(3); rotate3d on        % set 3-D view, turn on mouse-based 3-D rotation
```

3. Find the discriminant function for each class. Then, plot the discriminant functions for each class and the boundaries of the respective classes in the 2D feature space.

   Hints:

- The discriminant function of a class can be generated as follows:

```
% Find the discriminant function for a class
   discriminant(:, :, class) = p(:, :, class) * priori(class);
```

- The MATLAB function "**surfc**" or "**meshc**" can be used to plot the discriminant functions.

```
peak = max(max(max(discriminant)));      % find the maximum value of discriminant function
figure; hold on; grid on                 % hold on the current figure
                                         % add major grid lines in the graph
axis([range1, range2, -peak, peak])      % set the axes of the graph
for class = 1:numClass,                  % loop for each class
  surfc(x11, x22, discriminant(:, :, class)) % plot the discriminant functions
end
xlabel('x1 (Feature 1)'); ylabel('x2 (Feature 2)'); zlabel('g(x)')
title('Plot of Discriminant Functions of 2-D data')
hold off; view(3); rotate3d on           % set 3-D view, turn on mouse-based 3-D rotation
```

- A data point in **xx** is assigned to a specific class if the corresponding value of its discriminant function is a maximum when compared to that of other classes. You may use the MATLAB function "**contour**" to plot the boundaries. The following code may help you.

```
% Plot the boundaries of the classes
[maxDiscri, yy] = max(discriminant, [ ], 3);     % determine the choice 'yy'
figure; hold on                                  % hold the current figure on
contour(x11, x22, yy, numClass - 1)              % plot the boundaries first
plotdata2d(x, y);                                % plot the data points
hold off; axis([range1, range2])                 % hold the current figure off
xlabel('x1 (Feature 1)'); ylabel('x2 (Feature 2)')
title('Boundaries of 2-D data'); legend(['Class 1'; 'Class 2'; 'Class 3'; 'Class 4'])

% Plot the boundaries of the discriminant functions
figure; hold on; axis([range1, range2])
contour(x11, x22, yy, numClass - 1)              % plot the boundaries
for class = 1:numClass, % loop for each class
   contour(x11, x22, discriminant(:, :, class))  % plot the discriminant function
end
hold off
xlabel('x1 (Feature 1)'); ylabel('x2 (Feature 2)')
title('Boundaries of the 2-D discriminant functions')
```

4. Plot the decision boundaries between any two of the classes. The discriminant function $g(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i) - \frac{n}{2}\log 2\pi - \frac{1}{2}\log|\boldsymbol{\Sigma}_i| + \log P(\omega_i)$ will be implemented, where $\boldsymbol{x}$ is the data, and $\boldsymbol{\mu}_i$, Σ_i, and $P(\omega_i)$ are the mean, covariance matrix and prior probability of the ith class, respectively. Comment on the shape of the boundaries, and explain your answers.

   Hints:

   - Use the provided function “**plotboundary2d**” to plot the boundaries. This function is used to plot the boundaries of 2-D data of different classes, and also returns the figure handles. The discriminant functions are also implemented in this function. Note that this function uses some MATLAB functions in the symbolic library. If you don’t have this library, you may skip this part. The following codes may help you.

```
% Plot the boundaries for each class against other classes
figureHandle = plotboundary2d(u, C, numClass, priori, range1, range2);
for class = 1:numClass,               % loop for each class
   figure(figureHandle(class));       % make 'figureHandle(class)' to the current figure
   plotdata2d(x, y);                  % plot data in the graph
   xlabel('x1 (Feature 1)'); ylabel('x2 (Feature 2)')
   title(['Boundaries of 2-D data of class ', num2str(class)])
end
```

Appendix

“plotdata1d.m”

```
function dataRange = plotdata1d(x, y, numClass)
% PLOTDATA1D - Plot 1D data of N classes
%
% x - data point in the form of a row vector
% y - a row vector of the classes of the data
% numClass - number of classes
%
% This function returns 2x1 vector2s "dataRange",
% which is the approximate range of value of the data

% Set the range of data and correct it to 2 nearest significant figures
% The following function is developed at the end of this program
dataRange = setSigniRange(x);

if nargin < 3,                          % if the number of input is less than 3
   numClass = max(y);                   % set the number of classes to maximum value of y
end

hold on                                 % hold the current figure on
for n = 1:numClass,                     % loop for each class
   index = find (y == n);               % find the index of data belonging to class n
   % The following function is developed at the end of this program
   S = setColorMark(n);                 % set the color for this class
   stem(index, x(index), S)             % plot the data of this class
end
hold off                                % hold the current figure off
return

function S = setColorMark(n)
% SETCOLORMARK - set the mark or color to be used
% to present the point.
if n > 77,
   error(['The number of sets is larger than 77,'...
         ' which is beyond the limitation of this program!']);
end

color_table = ['k';'b';'g';'r';'c';'m';'y'];
mark_table = ['<';'o';'x';'s';'+';'*';'d';'^';'v';'p';'h'];
icolor = mod(n,7) + 1;
imark = mod(n,11) + 1;
S = [color_table(icolor), mark_table(imark)];
return

function dataRange = setSigniRange(x)
% SETSIGNIRANGE - set the range of data
% and correct it to 2 nearest significant figures
minValue = min(x);
maxValue = max(x);
if minValue == 0, minDigit = 0; % error if log10(0), so set “minDigit” to 0
else minDigit = floor(log10(abs(minValue))); end
if maxValue == 0, maxValue = 0; % error if log10(0), so set “maxDigit” to 0
else maxDigit = floor(log10(abs(maxValue))); end
Digit = max(minDigit, maxDigit) - 1; % choose the maximum signi. fig.
minValue = round(minValue / 10^Digit - 0.5) * 10^Digit;
maxValue = round(maxValue / 10^Digit + 0.5) * 10^Digit;
dataRange = [minValue, maxValue];
return
```

“**plotboundary1d.m**”

```
function plotboundary1d(xx, yy, peak)
% PLOTBOUNDARY1D - Plot the decision boundaries of 1-D data
%
% xx - data point, which should be a row vector
% yy - a row vector which indicates the classes of the data
% peak - a maximum value of discriminant function
temp = -0.1 * peak;                  % set the position for plotting the region line

hold on                              % hold on the current figure
previousClass = yy(1);               % for initialization
for n = 1:length(xx),                % loop for each data point
   S = setColor(yy(n));              % set the color for this class
   plot(xx(n), temp, S);             % plot the region line for this data

   if yy(n) ~= previousClass,        % detect the boundaries
      % suppose the boundary 'lie' at the midpoint
      mid = 0.5 * (xx(n-1) + xx(n));
      stem([mid, mid], [temp, peak], 'k:.'); % plot the boundaries
      previousClass = yy(n);         % update the variable 'previousClass'
   end
end
hold off                             % hold off the current figure
return

function S = setColor(n)             % SETCOLOR - set the color to be used to present the point.

color_table = ['k';'b';'g';'r';'c';'m';'y'];
icolor = mod(n,7) + 1;
S = [color_table(icolor), 'o'];
return
```

“**gendata.m**”

```
function [x, y, prior, label] = gendata(numDimension, numClass)
% GENDATA - Generate a data set with multiple dimensions & multiple classes
%
% numDimension - number of dimension of each data point
% numClass - number of classes
% x - data of size D x L, where D is number of dimension, L is the number of data point
% y - a row vector which indicates the classes of the points
% prior - the prior probabilities, which is a 1xN vector. N is the number of classes
% label - an Nx1 character array, store the label of classes

x = [ ]; y = [ ]; label= [ ];                          % for initialization
for class = 1:numClass,
   % generate number of points, mean, standard deviation randomly
   numPoints = 50 + ceil(rand * 20);                   % generate number of points randomly
   meanValue = ceil(rand(numDimension, 1) * 100);      % generate mean randomly
   stdValue = ceil(rand(numDimension,1) * 20);         % generate STD randomly
   xtemp = randn (numDimension, numPoints);            % generate points randomly
   ytemp = ones (1, numPoints) * class;                % indicate these points belonging this class
   for n = 1:numDimension,                             % loop for each dimension
      xtemp(n,:) = ...                         % justify all the random points to its mean and STD
         meanValue(n) + xtemp(n,:) .* stdValue(n);
   end
   % padding the data
   x = [x, xtemp];
   y = [y, ytemp];
   labeltemp = ['Class ', int2str(class)];
   label = [label; labeltemp];
end
prior = ones(1, numClass) / numClass;                  % generate the prior probabilities
return
```

"plotdata2d.m"

```
function [dataRange1, dataRange2]...
   = plotdata2d(x, y, numClass, range1, range2)
% PLOTDATA2D - Plot 2D data set with N Classes
% The maximum number of class is 77.
%
% x - data point, should be 2 x L in size, where L is number of data point
% y - a row vector which indicate which class the data is
% numClass - number of classes
% range1 - range of data on first dimension , e.g. [-10, 100]
% range2 - range of data on second dimension, e.g. [-10, 100]
% This function returns 2x1 vector2s "dataRange1", and "dataRange2",
% which are the approximate ranges of data on 1st and 2nd dimension,
% respectively.

% Set the range of data and correct it to 2 nearest significant figures
% The following function is developed at the end of this program.
dataRange1 = setSigniRange(x(1,:));
dataRange2 = setSigniRange(x(2,:));

if nargin < 5,                          % if the number of inputs is less than 5
   range2 = dataRange2;
end
if nargin < 4,                          % if the number of inputs is less than 4
   range1 = dataRange1;
end
if nargin < 3,                          % if the number of inputs is less than 3
   numClass = max(y);                   % set the number of classes to the maximum value of y
end

if numClass > 77,
   error(['The number of sets is larger than 77,'...
         ' which is beyond the limitation of this program!']);
   return
end

hold on                                 % hold the current figure on
axis([range1, range2]);                 % set the range of axis
for n = 1:numClass,                     % loop for each class
   index = find(y == n);                % find the indices of the data belonging to this class
   % The following function is developed at the end of this program.
   S = setColorMark(n); % set the color and mark for this class
   plot(x(1, index), x(2, index), S) % plot the data of this class
end
hold off                                % hold the current figure off

function S = setColorMark(n)
% SETCOLORMARK - set the mark or color to be used
% to present the point.
if n > 77,
   error(['The number of sets is larger than 77,'...
         ' which is beyond the limitation of this program!']);
   return
end

color_table = ['k';'b';'g';'r';'c';'m';'y'];
mark_table = ['<';'o';'x';'s';'+';'*';'d';'^';'v';'p';'h'];
icolor = mod(n,7) + 1;
imark = mod(n,11) + 1;
S = [color_table(icolor), mark_table(imark)];
return

function dataRange = setSigniRange(x)
% SETSIGNIRANGE - set the range of data
% and correct it to 2 nearest significant figures
minValue = min(x);
maxValue = max(x);
if minValue == 0, minDigit = 0; % error if log10(0), so set "minDigit" to 0
else minDigit = floor(log10(abs(minValue))); end
if maxValue == 0, maxValue = 0; % error if log10(0), so set "maxDigit" to 0
else maxDigit = floor(log10(abs(maxValue))); end
Digit = max(minDigit, maxDigit) - 1; % choose the maximum signi. fig.
minValue = round(minValue / 10^Digit - 0.5) * 10^Digit;
maxValue = round(maxValue / 10^Digit + 0.5) * 10^Digit;
dataRange = [minValue, maxValue];
return
```

“**plotboundary2d.m**”

```
function figureHandle = ...
   plotboundary2d(u, C, numClass, prior, range1, range2)
% PLOTBOUNDARY2D - Plot the boundaries for each class against other classes
%
% u - means of classes, should be a 2xL matrix, where there are L classes.
% C - Covariance matrix, should be a 2x2xL matrix, where there are L classes.
% numClass - number of classes
% prior - the prior probabilities of classes.
% range1 - range of data on 1st dimension, e.g. [-10, 100]
% range2 - range of data on 2nd dimension, e.g. [-10, 100]
% This function returns a 1xL vector, which contains the handles of the figures,
% L is the number of classes.

syms a b;                              % prepare the symbolic objects 'a' and 'b'
for class = 1:numClass,                % loop for each class
   eval(['g', num2str(class), ...      % implement the discriminant function
      '= -0.5 * transpose([a; b] - u(:,class))',...
      ' * inv(C(:,:,class)) * ([a; b] - u(:,class))',...
      ' - 0.5 * log(abs(det(C(:,:,class))))',...
      '+ log(prior(class));']);
end

hLength = length(findobj('Type', 'line'));   % for initialization
for class = 1:numClass,                % loop for each class
   figureHandle(class) = figure; hold on   % store the figure handle
   for otherClass = 1:numClass,        % loop for each class
      if otherClass == class,
         continue; % skip this step
      end
      eval(['g = g', num2str(class), '- g', num2str(otherClass),';']);
      if findsym(g) == 'a',            % if the equation remains the variable 'a'
         midpoint = eval('0.5 * (u(1,class) + u(1,otherClass))');
         ezplot(midpoint, 'a', [range1, range2])      % plot it as a vertical line
      elseif findsym(g) == 'b',        % if the equation remains the variable 'b'
         midpoint = eval('0.5 * (u(2,class) + u(2,otherClass))');
         ezplot(midpoint, [range2])    % plot it as a horizontal line
      else                             % if the equation remains both variables 'a' and 'b'
         ezplot(g, [range1, range2])   % plot the boundaries
      end
      h = findobj('Type', 'line');     % find the line handle
      set(h(1:length(h)-hLength), 'color', setColor(otherClass), ...
         'linestyle', ':');            % set the color of the line
      hLength = length(h);             % store the length of the line handle
   end
   hold off
end
return

function S = setColor(n)
```

```
% SETCOLOR - set the color to be used to present the point.

color_table = ['k';'b';'g';'r';'c';'m';'y'];
icolor = mod(n,7) + 1;
S = color_table(icolor);
return
```

– END –